

## Task A22. Cycles

 1 s  256 MB

Yan is a brave boy who always takes the shortest path from school to his grandmother's house. Unfortunately, this path leads directly through the haunted forest.

One day, while walking through the forest, Yan encounters the terrifying creature Tung Tung Sahur. The creature challenges him to a game: if Yan wins, he will be allowed to continue on his way - and may even receive a reward. If he loses, however, Tung Tung Sahur will eat him.

The game begins with Tung Tung Sahur choosing a hidden permutation  $P$  with  $N$  elements. Now it's Yan's turn to play. He may repeatedly choose two different positions in the permutation and ask Tung Tung Sahur to swap the elements at those positions. Tung Tung Sahur will carry out the swap and then announce the number of **cycles**<sup>1</sup> in the updated permutation. These swaps are persistent - Tung Tung Sahur does not revert them (but Yan is free to repeat a swap to revert it if needed).

At any point, Yan may declare that the permutation is sorted by shouting "Sorted." If the permutation is indeed sorted in increasing order, Tung Tung Sahur lets him go. Moreover, Tung Tung Sahur will reward Yan with some gold coins, the amount depending on how few swaps were needed. Therefore, Yan must try to sort the permutation using as few swaps as possible.

Your task is to help Yan sort the hidden permutation using only the given information, and try to do so while minimizing the number of swaps needed.

### Implementation details

You have to implement the function:

```
void sortPermutation(int N);
```

It will be called once per test case with  $N$  - the number of elements in the permutation Yan needs to sort. From this function (and other functions you write), you can call the function `performSwap` to swap 2 elements of the permutation, and will in turn receive the total number of cycles in the resulting permutation. You must guarantee that after your function finishes, the hidden permutation is sorted in increasing order  $P_i = i$  for all  $0 \leq i < N$ , since Yan will immediately shout "Sorted" after this function returns.

```
int performSwap(int x, int y);
```

Calling `performSwap` represents swapping the element  $P_x$  with  $P_y$ . Note that, if you call this function with the same element, i.e.  $x = y$ , you will receive a Wrong Answer verdict. This function receives  $x$  and  $y$  as 0-based indices and returns the number of cycles in the **updated** permutation.

---

<sup>1</sup>Any permutation of the integers  $\{0, 1, \dots, N-1\}$  can be decomposed into a collection of disjoint cycles. To find one such cycle, begin at any index  $i$ , and follow the mapping  $i \mapsto P_i \mapsto P_{P_i} \mapsto \dots$  until you return to  $i$  - these form one cycle. For example, the permutation  $[1, 2, 0, 5, 4, 3]$  represents the mapping  $0 \mapsto 1, 1 \mapsto 2, 2 \mapsto 0, 3 \mapsto 5, 4 \mapsto 4$  and  $5 \mapsto 3$ , and resulting in disjoint cycles  $(0\ 1\ 2)$ ,  $(3\ 5)$  and  $(4)$ , so it has 3 cycles.

## XVI INTERNATIONAL ADVANCED TOURNAMENT IN INFORMATICS SHUMEN 2025

Your code will be compiled together with a grader, so it should not implement a `main` function, nor should you read from `stdin` or write to `stdout`. You also need to include the header `cycles.h`. The permutation is fixed before calling `sortPermutation` and so the grader is not adaptive.

### Local testing

To test your program locally, a local grader and a header file are provided. The local grader first reads  $N$  and then  $N$  distinct numbers from 0 to  $N - 1$ . After this it calls your `sortPermutation` function and expects it to correctly sort the permutation.

### Constraints

- $N = 1000$

### Subtasks

Subtask	Points	Constraint	Additional constraints
1	10	$N = 1000$	For even $i$ : $(P_i, P_{i+1}) = (i, i + 1)$ or $(i + 1, i)$ .
2	20	$N = 1000$	The initial permutation can be sorted by performing exactly one swap.
3	70	$N = 1000$	–

You get a fraction of the points for a given subtask, only if you correctly pass all tests in it. Your result will be calculated based on  $Q$  – the maximum number of calls to `performSwap` among all those tests:

$10^7 < Q$	0%
$10^6 < Q \leq 10^7$	10%
$9 \cdot 10^4 < Q \leq 10^6$	20%
$3 \cdot 10^4 < Q \leq 9 \cdot 10^4$	60%
$Q \leq 3 \cdot 10^4$	100%

### Sample test

Input	Interaction
3 2 0 1	<code>sortPermutation(3)</code> <code>performSwap(0, 1): return 2</code> <code>performSwap(0, 1): return 1</code> <code>performSwap(0, 1): return 2</code> <code>performSwap(1, 2): return 3</code> <code>sortPermutation(3): returns</code>